C1 said basic block for a given program so that the execution of said branch instruction occurs in parallel with the execution of said non-branch instructions in said basic block whereby overall processing of all said programs is increased.

In the Title

Change the title of this application to "Method and Apparatus

C2 for Executing Branch Instructions".

REMARKS

Reexamination and reconsideration of the claims, as amended, are respectfully requested.

Applicant's attorney apologizes for failing to provide pages 56 and 57 of the June 13, 1988 amendment in the grandparent application when the first Preliminary Amendment was filed in this case. Those pages are attached hereto.

Applicant is also providing drawings wherein the changes have been shown in red.

Claims 69-70 have been cancelled and it is the present intention to file these claims in a separate divisional application.

As requested by the Examiner, I am enclosing a copy of the Information Disclosure Statement (Petition Pursuant to 37 C.F.R. 1.102(d) Advancement of Examination) filed in connection with the grandparent application.

Since the filing of this application, and during the review of the various files relating to the subject matter of this invention, I have found a memorandum which indicates that Gordon Morrison is the sole inventor of the claims pending in this divisional application. Accordingly, we respectfully request that Messrs. Brooks and Gluck be removed as inventors in this case.

The claims now pending in this application have been rejected under 35 U.S.C. §112 as being indefinite for failing to particularly point out and distinctly claim the subject matter of the invention. The claims have been amended to meet the objection that they are narrative, indefinite, and use functional or operational language. In addition, applicant has responded to the Examiner's objections regarding the use of numerical reference numbers in the claims and various specific other language.

The claims have also been rejected for failing to meet the non-obvious requirement of 35 U.S.C. §103 over the McDowell references in view of Freiman et al. It is respectfully submitted that the claims clearly define patentable subject matter over the McDowell references and Freiman et al for the following reasons.

While the bulk of the specification herein, which derives from the grandparent application, is directed to a parallel processing computer, the invention claimed in this divisional application is directed specifically to a method and apparatus for executing branches and can be used in simpler machines wherein a parallel structure can be implemented specifically directed to executing the branch instruction.

Referring to the McDowell references, there is no suggestion, teaching, or discussion with regard to early processing of a branch instruction so that it, for example, executes beginning at a time earlier than the last non-branch instruction of a block of instructions. To our knowledge, this approach to branch instruction execution was not implemented in any machine commercially introduced during the 1980's or earlier. Further, McDowell, for example in his thesis, in Figures 6.2, 6.3, and 6.4, illustrates only the step of placing the branch instruction at the end of the basis block processing stream, along with the last non-branch instruction to be executed. Thus the branch instruction, which can require more time to execute than, for example, a simple non-branch instruction, especially when it is a conditional branch, may not complete execution during the execution of the last non-branch instruction. If the branch and non-branch instructions complete execution at the same time (or if the branch completes earlier), there is no "cost" associated with executing the branch. McDowell is silent on the execution of the branch instruction and does not address the problem or provide the solution described and claimed in this application.

The Freiman et al reference refers, at column 3, lines 1-5, to a concept which, as the Examiner notes, suggests determining the latest times during which an instruction can be performed, and then optimizing the utilization of the processors to achieve a best performance. Freiman et al, however, do not refer to the branch instruction, and do not recognize that the branch

instruction can be processed in parallel with the other instructions being processed. This is true even though the concept of trying to process two instructions at the same time was well known.

Further, the Freiman et al patent is directed to increasing processing speed for evaluating an algebraic equation. Branch instructions are not mentioned in Freiman et al because they are not considered. The focus in Freiman et al is to provide earliest and latest operating times of the arithmetic operations which would be used to evaluate an algebraic expression. Once those times are provided, the sequence of steps for the various processes which are available can be determined. Accordingly, the operations of Freiman et al are not branch operations, and Freiman et al provide no help with regard to how branch instructions should be handled. Indeed, in the 18 years between the grant of the Freiman et al patent and the filing of the grandparent application to the present application, no one has solved the branch instruction execution problem except the inventor herein.

It is thus the unique nature of the branch instruction and the various kinds of branch instruction which might be available, conditional branches, absolute branches, etc., which make branch instruction a unique and different. Accordingly, neither McDowell nor Freiman et al teach or suggest the claimed invention. The parallel process aspect of McDowell adds no additional feature or advantage here, since if Freiman et al had taught the claimed

invention, it would be useful not only in McDowell but substantially all types of parallel processors.

For the reasons noted above, it is respectfully submitted that the application, as amended, should be passed to issue in due course.

Respectfully submitted,

Gary A. Walpert
Reg. No. 26,098

Hale and Dorr
60 State Street
Boston, MA 02109
617/742-9100

October 4, 1991

Claims 74-77 have been added to more clearly and completely protect the invention herein. It is respectfully submitted that each of these claims belongs in the group of claims now being examined and that the claims should be allowed for one or more of the reasons noted above with regard to the claims presently pending in the application. In particular, for example, claim 74 provides an architecture which includes the full interconnection of the logic resource drivers, the processor elements, and the shared storage resources. None of the references provide the total computer architecture called for by the claim, and, in particular, the logical resource driver structure as described in the present application. As the Examiner will understand, each of claims 74-77 added herein tends generally to be broader than a corresponding claim in the application as originally filed. However, a review of the prior art indicates that such breadth of coverage should be allowable here since the references simply do not provide a teaching, suggestion, or description of the overall architecture or operation of the claimed parallel processing computer system.

I am further enclosing herewith a request for amendment of the specification using a substitute specification. The application has been reviewed in great detail, in part in connection with a foreign filing effort; and as a result, numerous typographical errors have been found and corrected. In addition, the description of the prior art herein has been deleted since it appears in similar form in applicants' petition and is in any case

of record in connection with the application. It is not considered necessary to add these twenty or more pages of text to the application itself. Furthermore, certain obvious errors, which are clear from a reading and understanding of the application as filed, have been corrected to avoid the necessity of a later reader having to perform the same ordinary corrective functions and to provide thereby a clearer and more accurate description of the invention. No new matter has been added. To aid the Examiner in his review of the substitute specification, I am enclosing herewith a so-called "blacklined" version of the application wherein the new language has been underlined and the deleted language has been enclosed in brackets. For the reasons noted above, it is respectfully submitted that the substitute specification should be entered.

For the reasons noted above, it is respectfully submitted that the objections under 35 USC §112 and the rejection based upon 35 USC §103 have been cured and that the application should be passed to issue in due course.

Respectfully submitted,

HALE AND DORR

Gary A. Walpert
Reg. No. 26,098
Attorney for Applicants

60 State Street
Boston, MA 02109
617/742-9100

June 13, 1988

Part of Paper # 6/C

DRAWINGS MARKED IN RED TO SHOW CHANGES
U.S. Serial No. 560,093

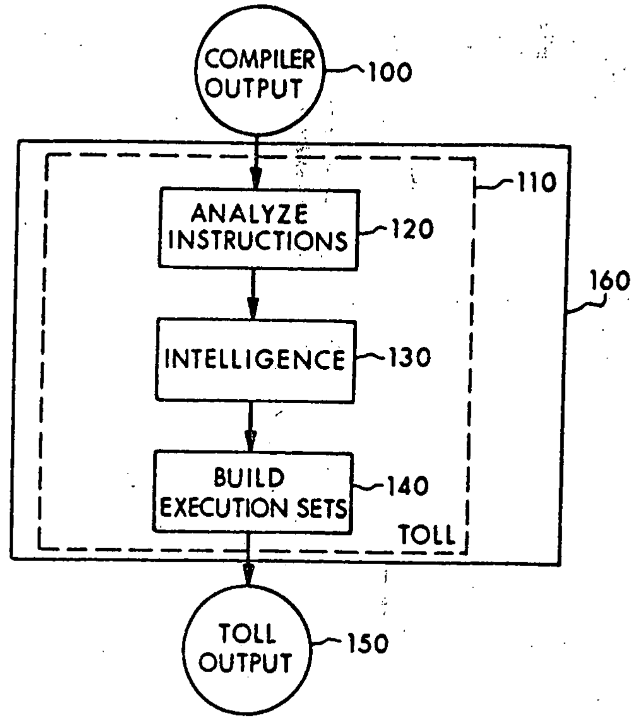
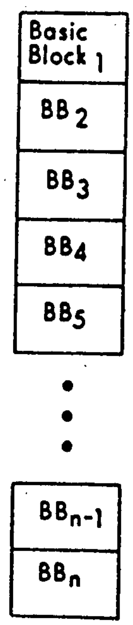


Fig. 1

Fig. 2
Prior Art



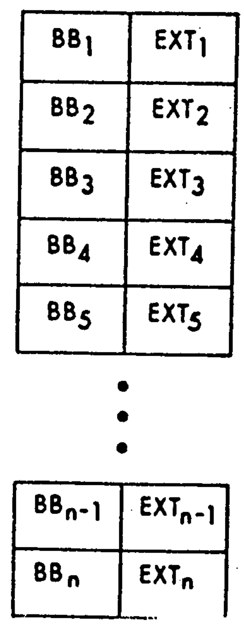


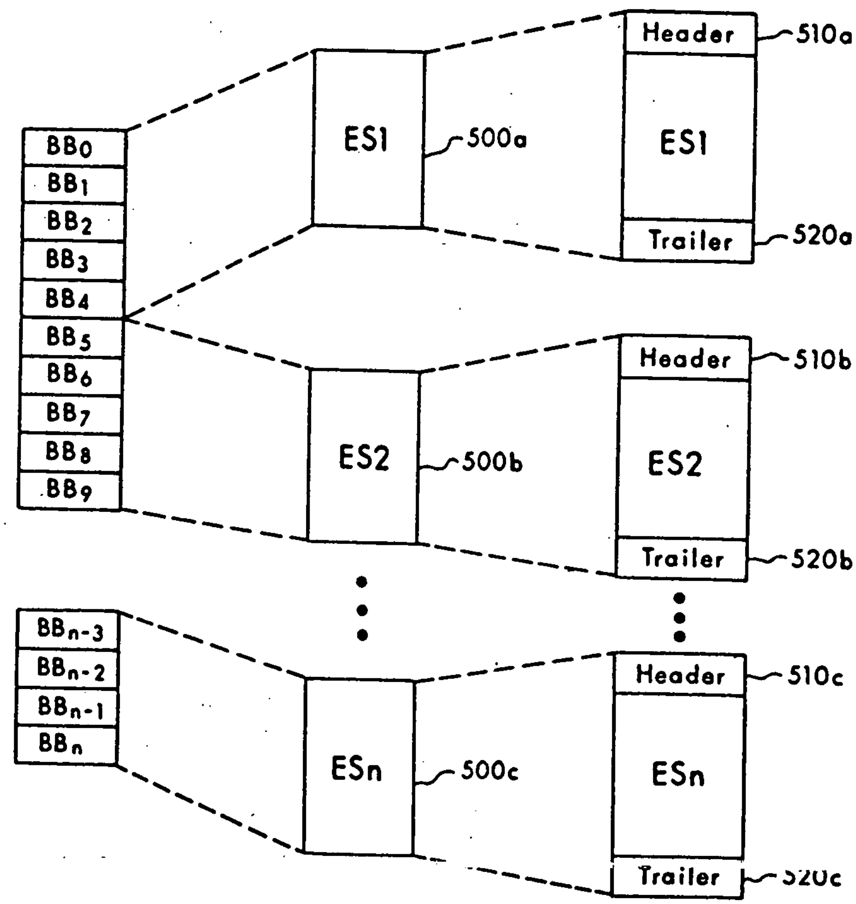
Fig. 3

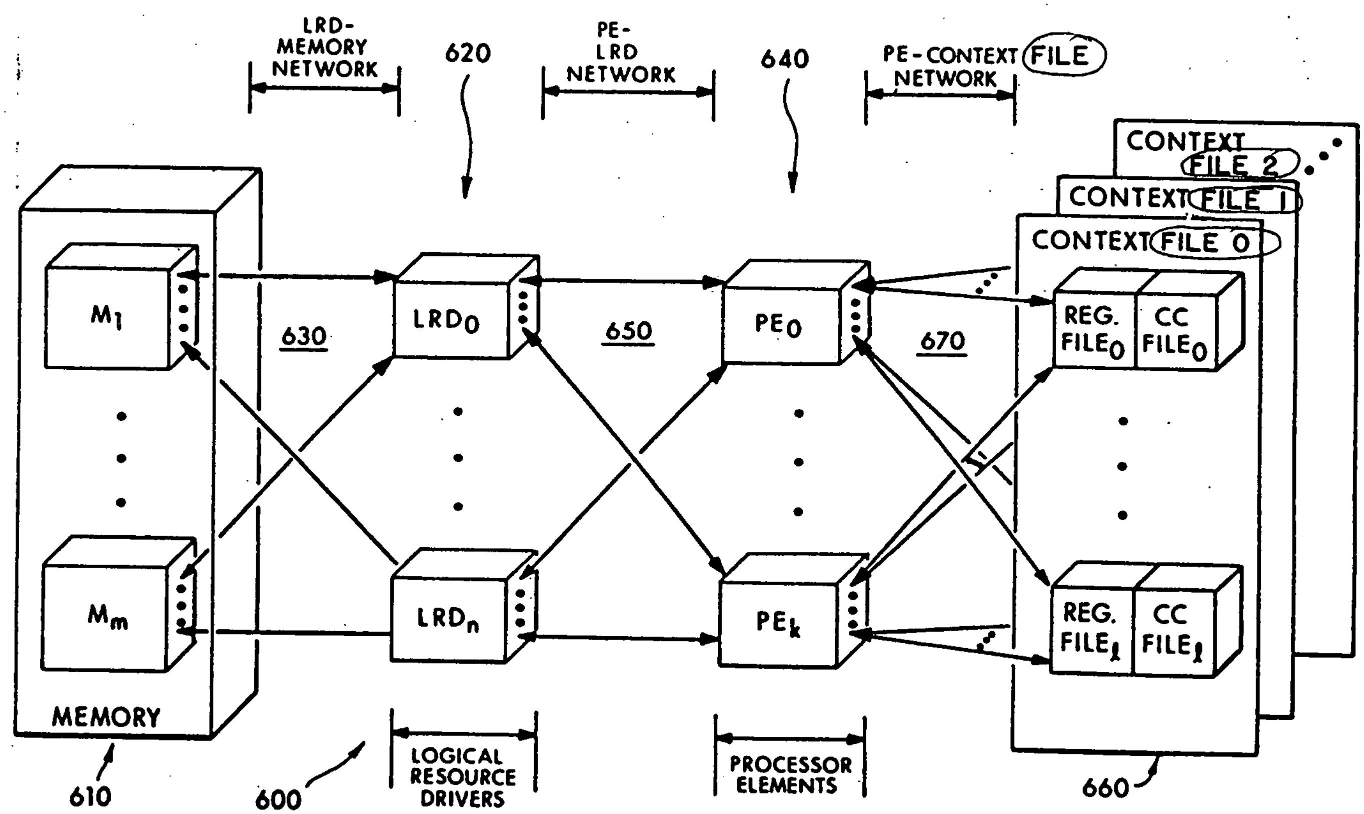
07/560,093

Fig. 4

I0	LPN_0	IFT_0	$SCSM_0$
I1	LPN_1	IFT_1	$SCSM_1$
⋮			
In	LPN_n	IFT_n	$SCSM_n$

Fig. 5





LRD-
MEMORY
NETWORK
620
PE-
LRD
NETWORK
640
PE-CONTEXT FILE
NETWORK
CONTEXT FILE 2
CONTEXT FILE 1
CONTEXT FILE 0
M1
Mm
MEMORY
630
LRD0
LRDn
650
PE0
PEk
670
REG. FILE0
CC FILE0
REG. FILEl
CC FILEl
610
600
LOGICAL
RESOURCE
DRIVERS
PROCESSOR
ELEMENTS
660

Fig. 6

09/566,093

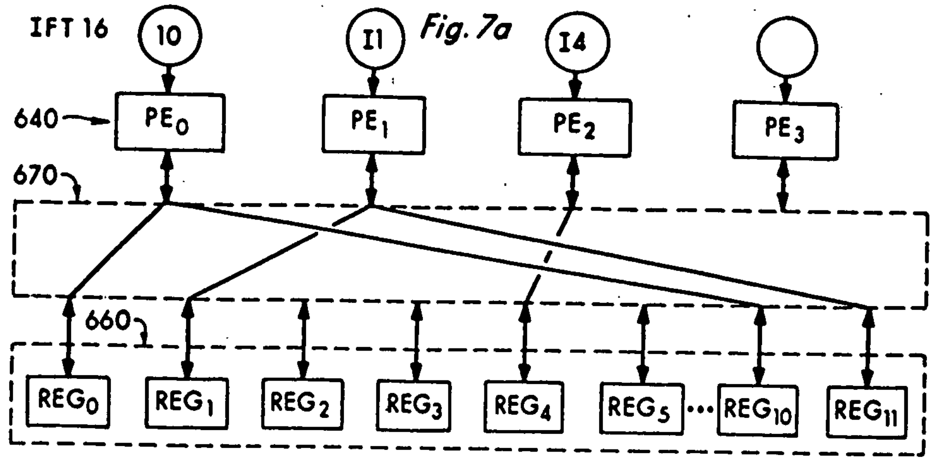


Fig. 7a

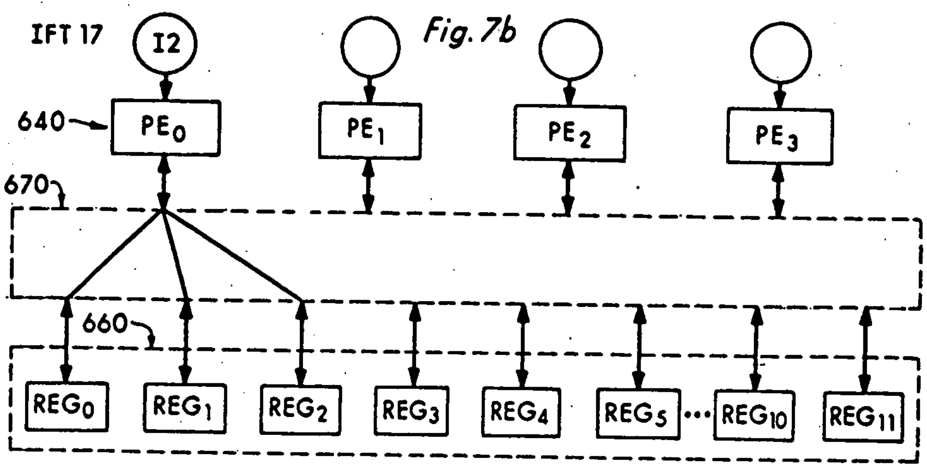


Fig. 7b

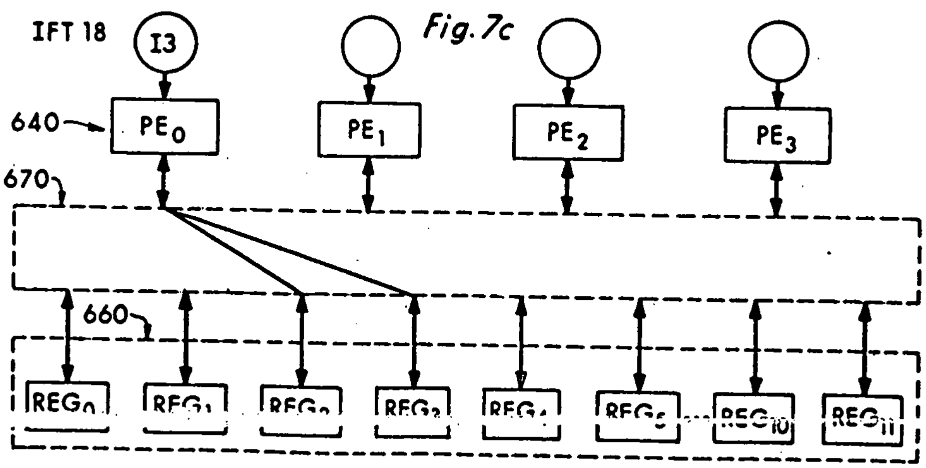


Fig. 7c

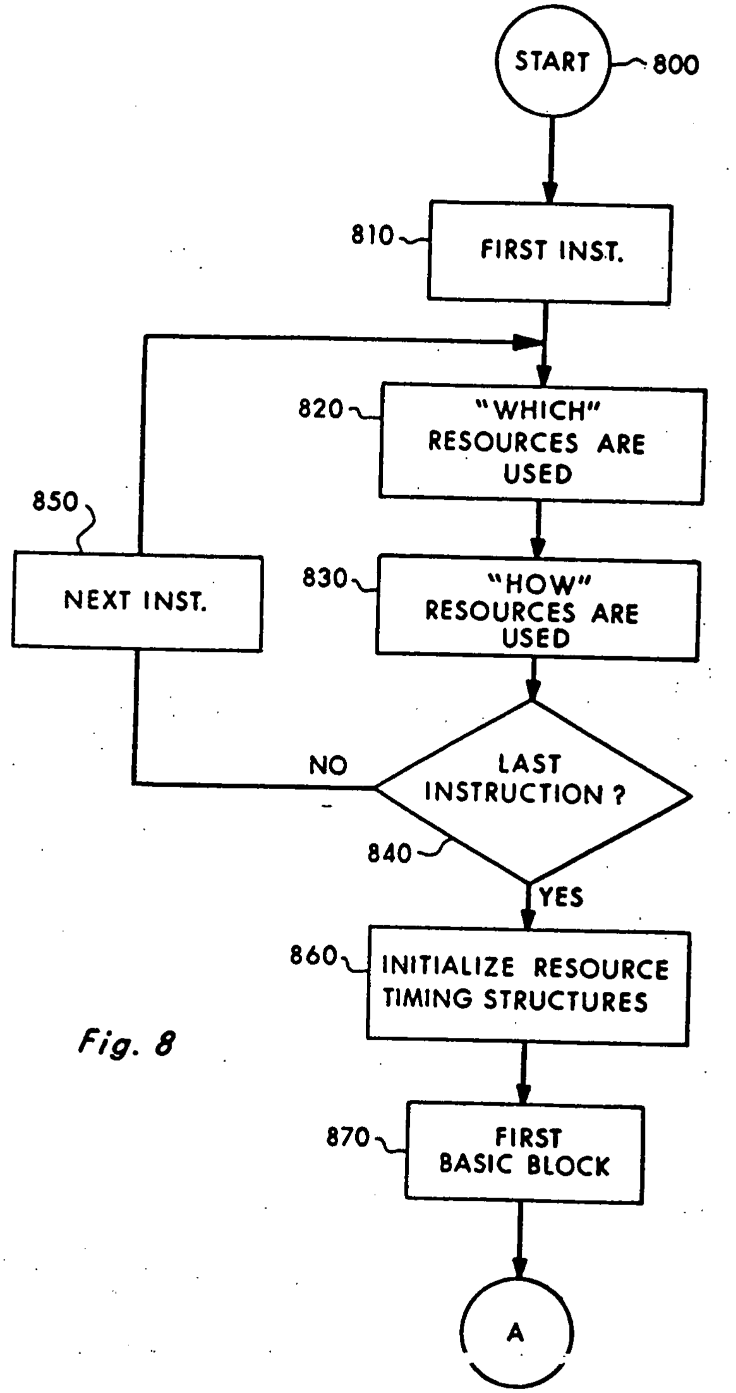

START
800
810
FIRST INST.
820
"WHICH" RESOURCES ARE USED
850
NEXT INST.
830
"HOW" RESOURCES ARE USED
NO
LAST INSTRUCTION ?
840
YES
860
INITIALIZE RESOURCE TIMING STRUCTURES
870
FIRST BASIC BLOCK
A

Fig. 8

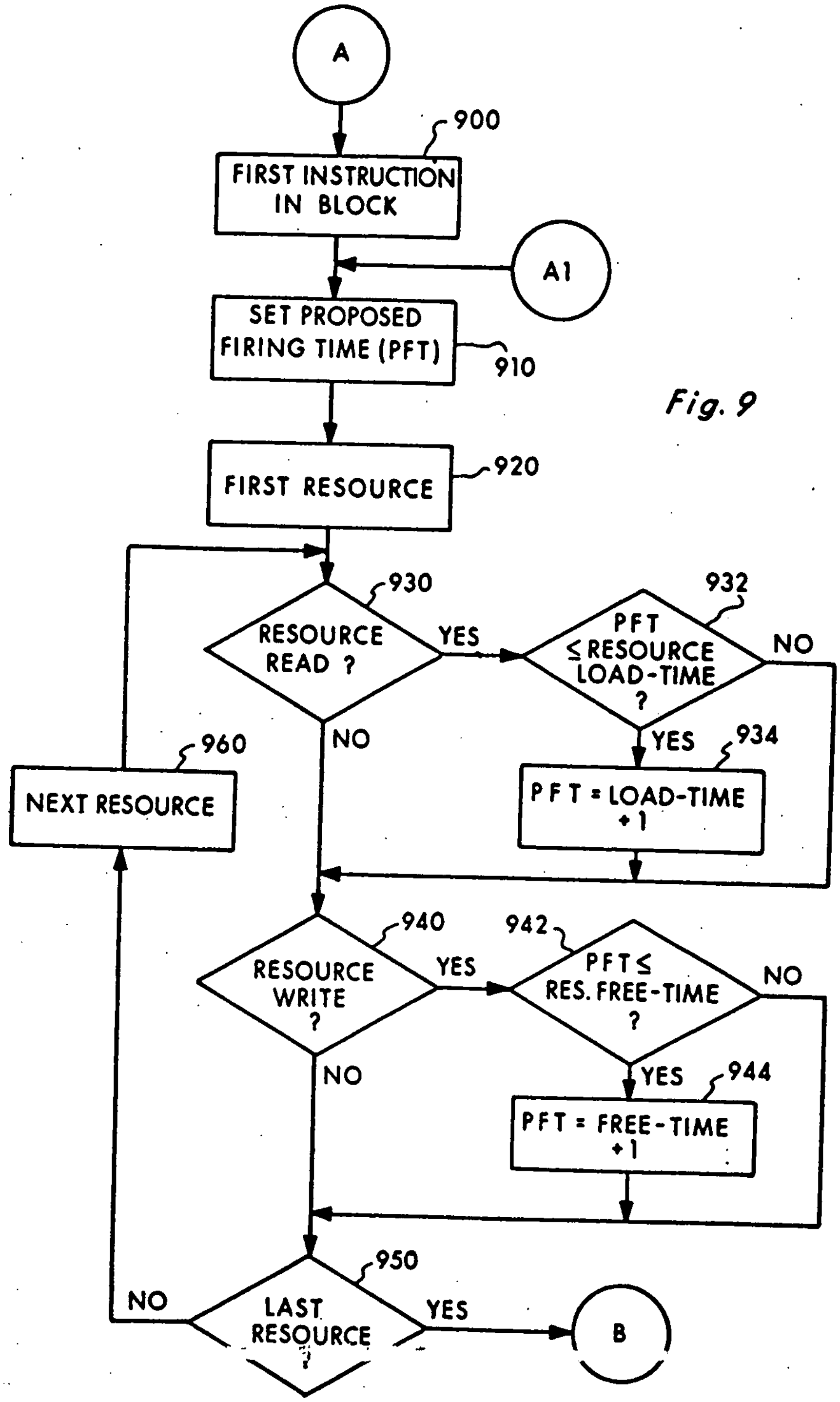
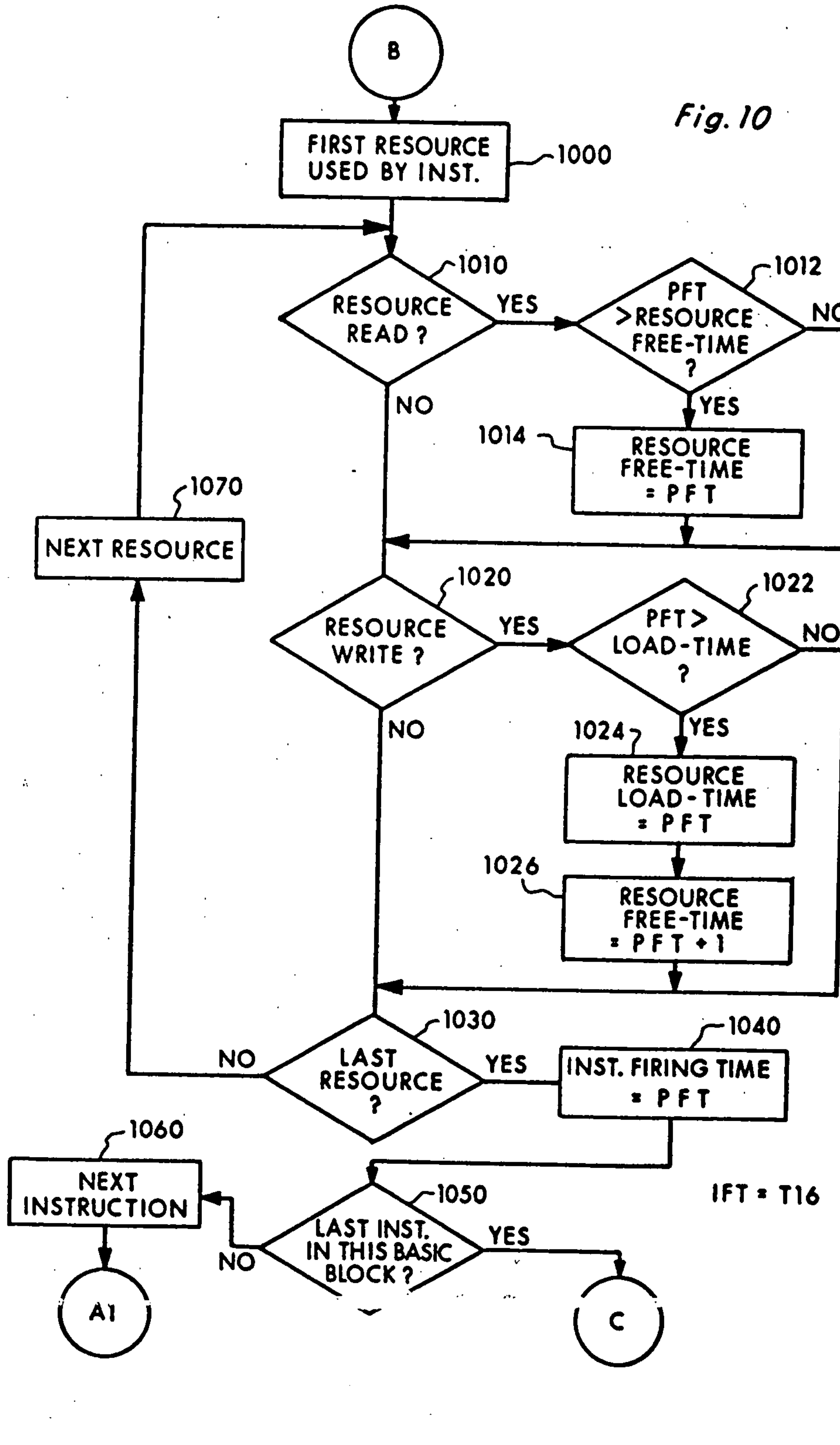

A
900
FIRST INSTRUCTION IN BLOCK
A1
SET PROPOSED FIRING TIME (PFT)
910
FIRST RESOURCE
920
930
RESOURCE READ ?
YES
932
PFT ≤ RESOURCE LOAD-TIME ?
NO
YES
934
PFT = LOAD-TIME +1
NO
960
NEXT RESOURCE
940
RESOURCE WRITE ?
YES
942
PFT ≤ RES. FREE-TIME ?
NO
YES
944
PFT = FREE-TIME +1
NO
950
LAST RESOURCE ?
NO
YES
B

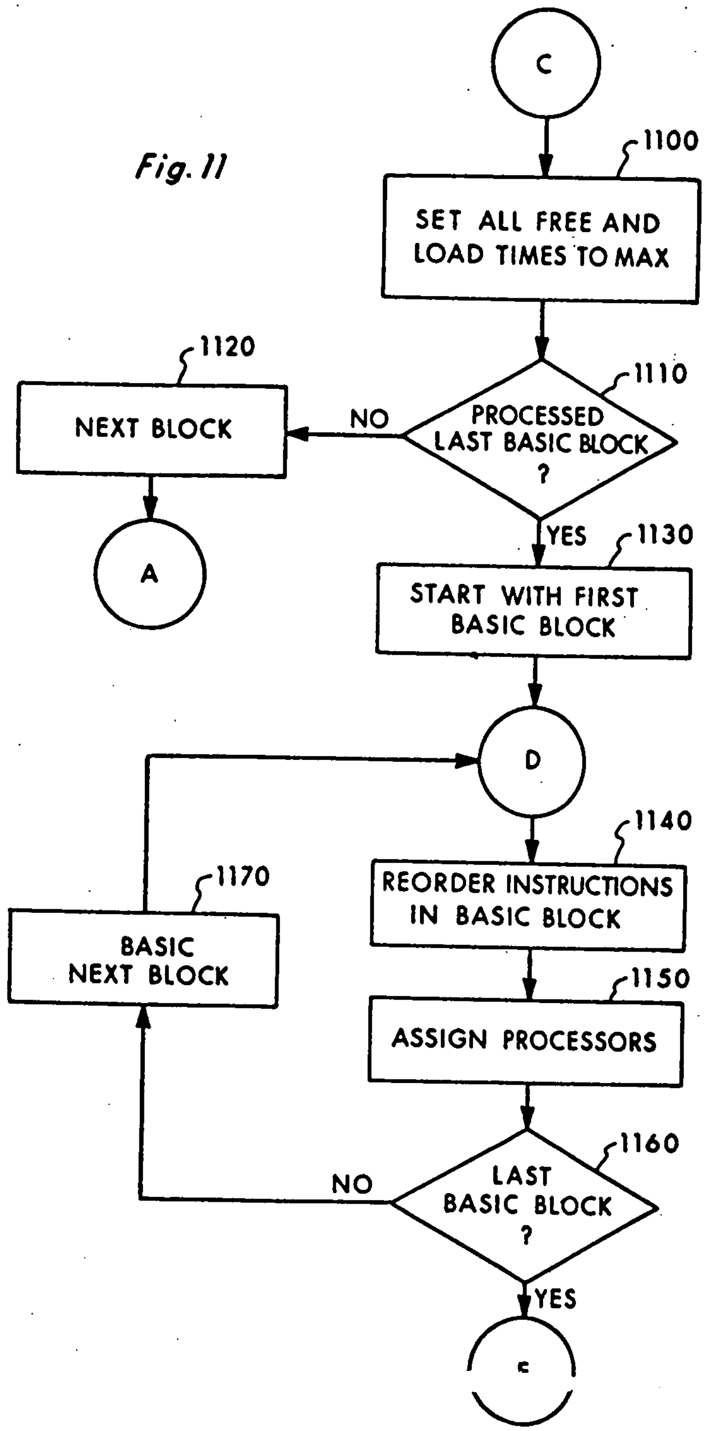
Fig. 9

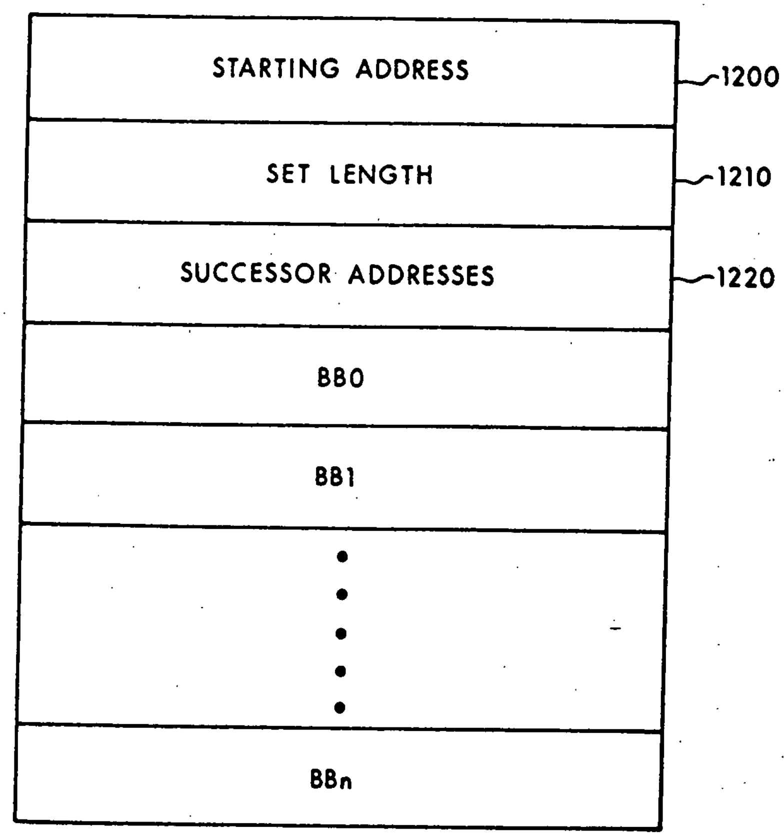


B
FIRST RESOURCE USED BY INST.
1000
RESOURCE READ ?
1010
YES
NO
PFT >RESOURCE FREE-TIME ?
1012
NO
YES
1014
RESOURCE FREE-TIME = PFT
1070
NEXT RESOURCE
RESOURCE WRITE ?
1020
YES
NO
PFT > LOAD-TIME ?
1022
NO
YES
1024
RESOURCE LOAD-TIME = PFT
1026
RESOURCE FREE-TIME = PFT + 1
LAST RESOURCE ?
1030
NO
YES
INST. FIRING TIME = PFT
1040
1060
NEXT INSTRUCTION
LAST INST. IN THIS BASIC BLOCK ?
1050
NO
YES
A1
C
IFT = T16

Fig. 10

Fig. 11


C
1100
SET ALL FREE AND LOAD TIMES TO MAX
1110
PROCESSED LAST BASIC BLOCK ?
NO
1120
NEXT BLOCK
A
YES
1130
START WITH FIRST BASIC BLOCK
D
1140
REORDER INSTRUCTIONS IN BASIC BLOCK
1150
ASSIGN PROCESSORS
1160
LAST BASIC BLOCK ?
NO
1170
BASIC NEXT BLOCK
YES



STARTING ADDRESS
1200
SET LENGTH
1210
SUCCESSOR ADDRESSES
1220
BB0
BB1
BBn

Fig. 12

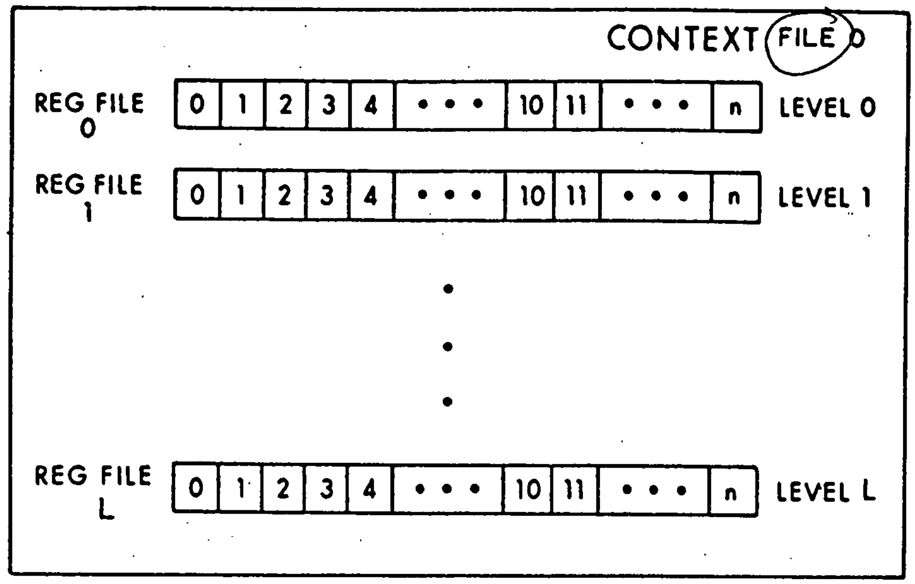

CONTEXT FILE 0
REG FILE 0
0 1 2 3 4 • • • 10 11 • • • n
LEVEL 0
REG FILE 1
0 1 2 3 4 • • • 10 11 • • • n
LEVEL 1
REG FILE L
0 1 2 3 4 • • • 10 11 • • • n
LEVEL L

Fig. 13

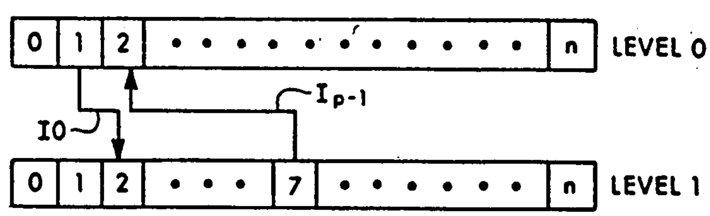
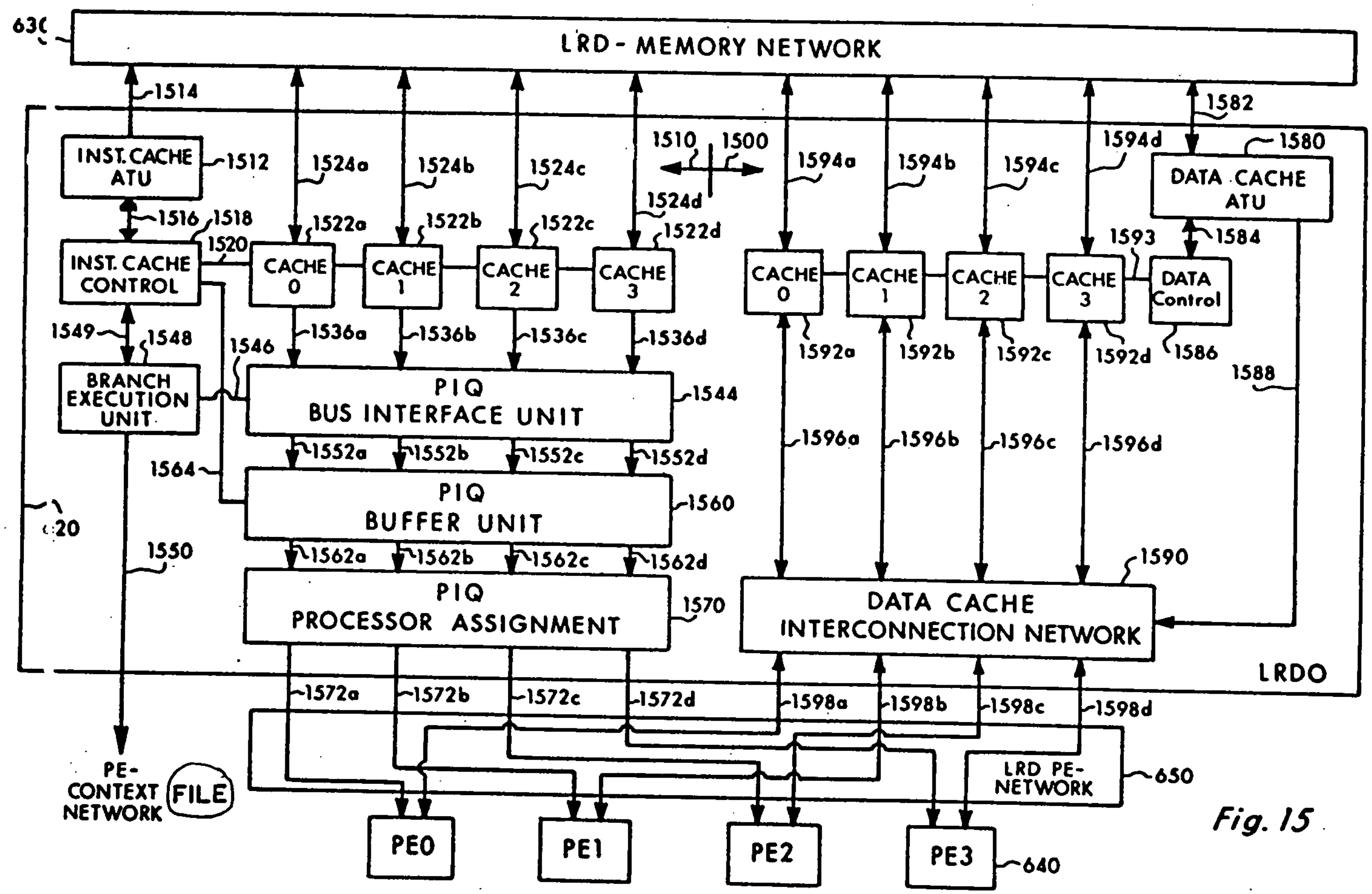
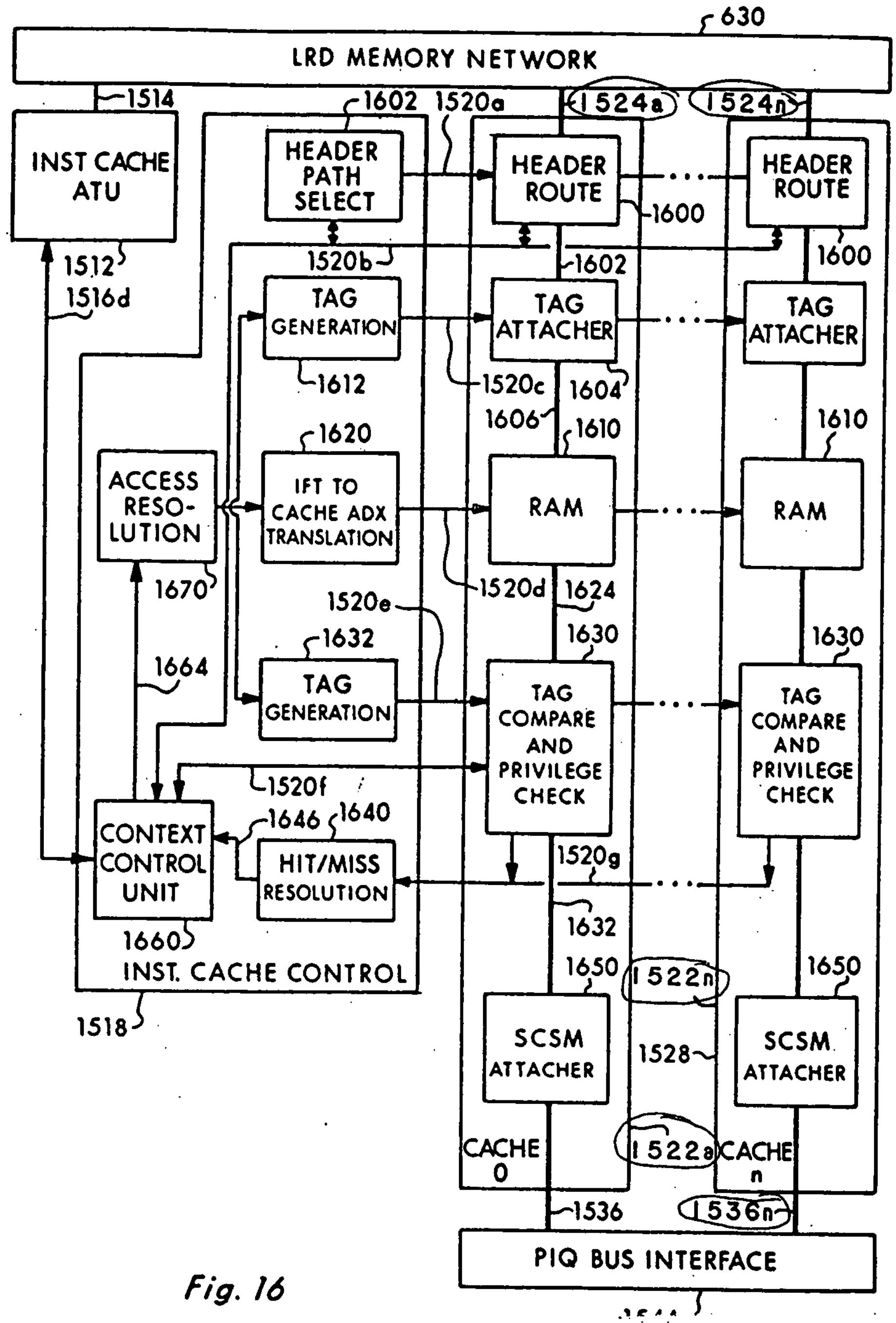

0 1 2 n
LEVEL 0
I_{p-1}
I0
0 1 2 • • • 7 n
LEVEL 1

Fig. 14



630
LRD - MEMORY NETWORK
1514
INST. CACHE ATU
1512
1516
1518
1520
INST. CACHE CONTROL
1549
1548
BRANCH EXECUTION UNIT
1546
1564
620
1550
PE-CONTEXT NETWORK
FILE
1524a
1524b
1524c
1524d
1522a
1522b
1522c
1522d
CACHE 0
CACHE 1
CACHE 2
CACHE 3
1536a
1536b
1536c
1536d
PIQ BUS INTERFACE UNIT
1544
1552a
1552b
1552c
1552d
PIQ BUFFER UNIT
1560
1562a
1562b
1562c
1562d
PIQ PROCESSOR ASSIGNMENT
1570
1572a
1572b
1572c
1572d
1510
1500
1594a
1594b
1594c
1594d
1582
1580
DATA CACHE ATU
1584
1593
DATA Control
1586
1592a
1592b
1592c
1592d
1588
1596a
1596b
1596c
1596d
1590
DATA CACHE INTERCONNECTION NETWORK
LRDO
1598a
1598b
1598c
1598d
LRD PE-NETWORK
650
PE0
PE1
PE2
PE3
640

Fig. 15



630
LRD MEMORY NETWORK
1514
INST CACHE ATU
1512
1516d
1602
1520a
1524a
1524n
HEADER PATH SELECT
HEADER ROUTE
HEADER ROUTE
1600
1520b
1602
1600
TAG GENERATION
TAG ATTACHER
TAG ATTACHER
1612
1520c
1604
1606
1610
1610
1620
ACCESS RESO-LUTION
IFT TO CACHE ADX TRANSLATION
RAM
RAM
1670
1520d
1624
1520e
1630
1630
1632
1664
TAG GENERATION
TAG COMPARE AND PRIVILEGE CHECK
TAG COMPARE AND PRIVILEGE CHECK
1520f
1646
1640
CONTEXT CONTROL UNIT
HIT/MISS RESOLUTION
1520g
1660
1632
INST. CACHE CONTROL
1650
1522n
1650
1518
SCSM ATTACHER
1528
SCSM ATTACHER
CACHE 0
1522a
CACHE n
1536
1536n
PIQ BUS INTERFACE

Fig. 16

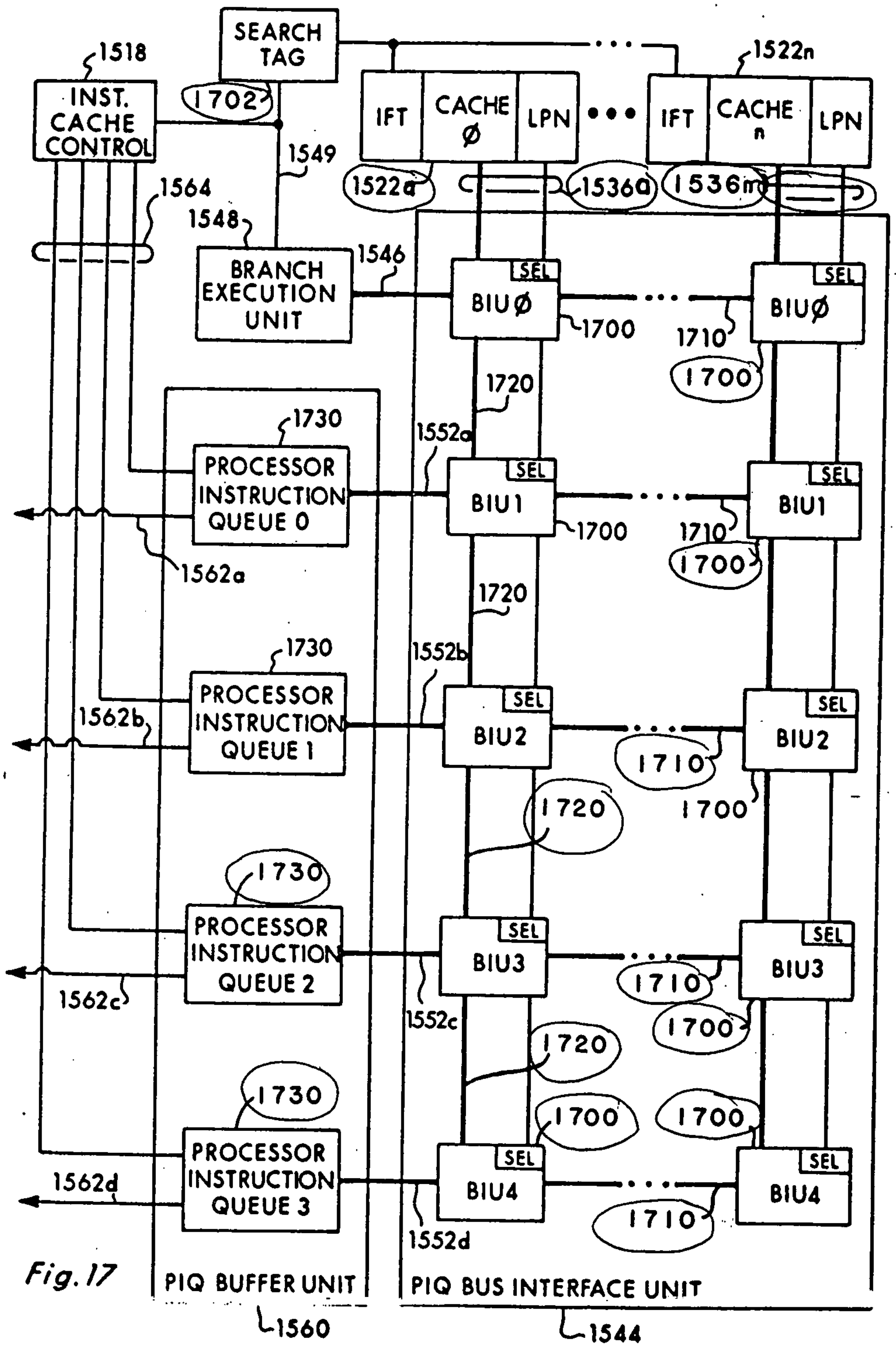
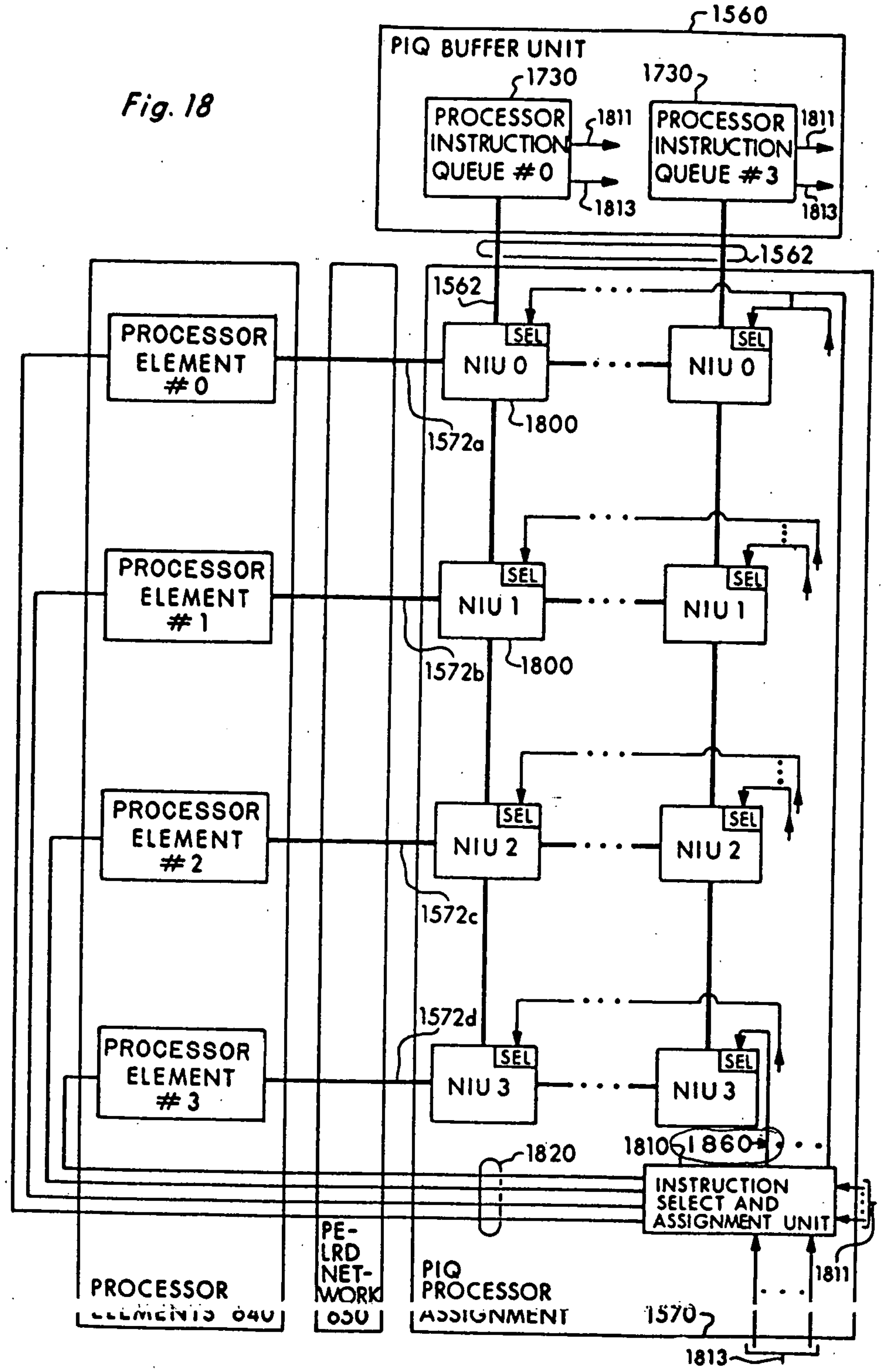

SEARCH TAG
1518
INST. CACHE CONTROL
1702
1522n
IFT
CACHE Ø
LPN
IFT
CACHE n
LPN
1549
1522a
1536a
1536n
1564
1548
BRANCH EXECUTION UNIT
1546
SEL
BIUØ
1700
1710
1720
1730
1552a
PROCESSOR INSTRUCTION QUEUE 0
BIU1
1562a
1552b
PROCESSOR INSTRUCTION QUEUE 1
1562b
BIU2
PROCESSOR INSTRUCTION QUEUE 2
BIU3
1562c
1552c
PROCESSOR INSTRUCTION QUEUE 3
BIU4
1562d
1552d
PIQ BUFFER UNIT
PIQ BUS INTERFACE UNIT
1560
1544
Fig. 17

Fig. 18



1560
PIQ BUFFER UNIT
1730
1730
PROCESSOR INSTRUCTION QUEUE #0
PROCESSOR INSTRUCTION QUEUE #3
1811
1811
1813
1813
1562
1562
PROCESSOR ELEMENT #0
PROCESSOR ELEMENT #1
PROCESSOR ELEMENT #2
PROCESSOR ELEMENT #3
SEL
NIU 0
NIU 0
NIU 1
NIU 1
NIU 2
NIU 2
NIU 3
NIU 3
1800
1800
1572a
1572b
1572c
1572d
1810
1860
1820
INSTRUCTION SELECT AND ASSIGNMENT UNIT
1811
1813
1570
PROCESSOR ELEMENTS 640
PE-LRD NETWORK 650
PIQ PROCESSOR ASSIGNMENT

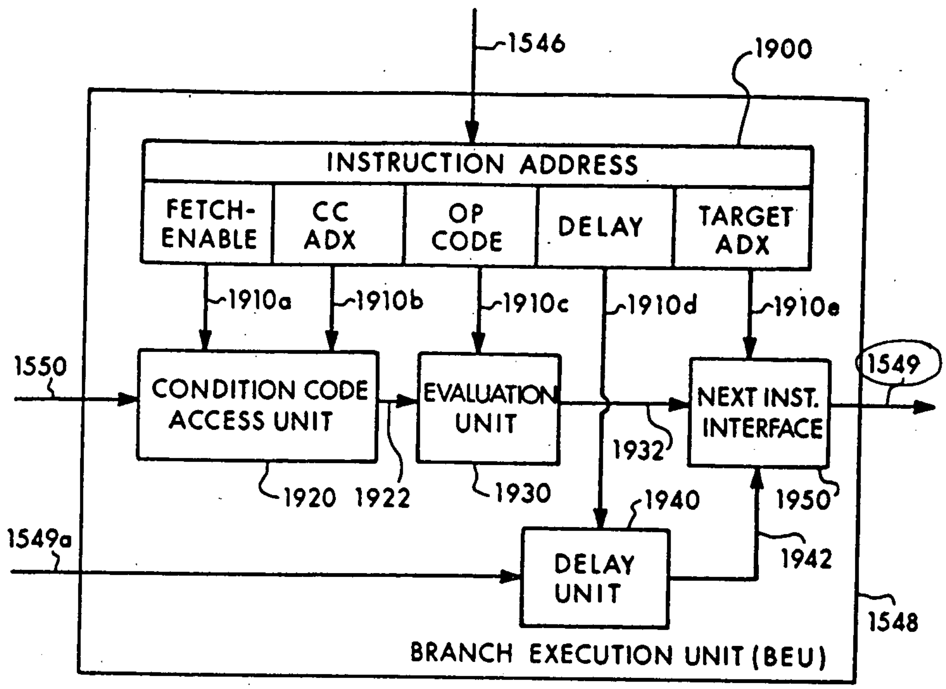

1546
1900
INSTRUCTION ADDRESS
FETCH-ENABLE
CC ADX
OP CODE
DELAY
TARGET ADX
1910a
1910b
1910c
1910d
1910e
1550
CONDITION CODE ACCESS UNIT
EVALUATION UNIT
NEXT INST. INTERFACE
1549
1932
1920
1922
1930
1940
1950
1549a
DELAY UNIT
1942
1548
BRANCH EXECUTION UNIT (BEU)

Fig. 19

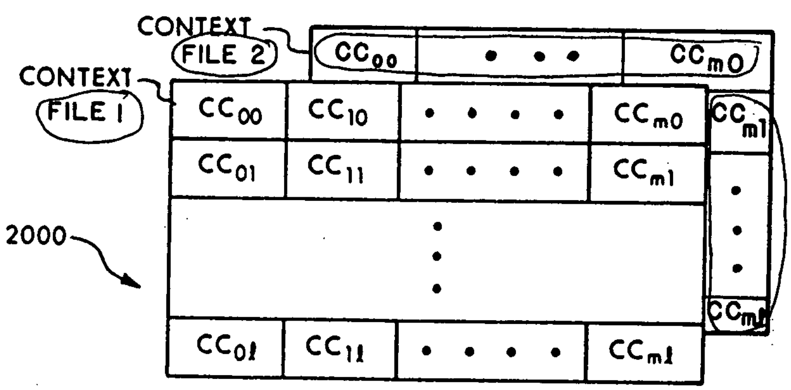

CONTEXT FILE 2
CONTEXT FILE 1
2000

Fig. 20

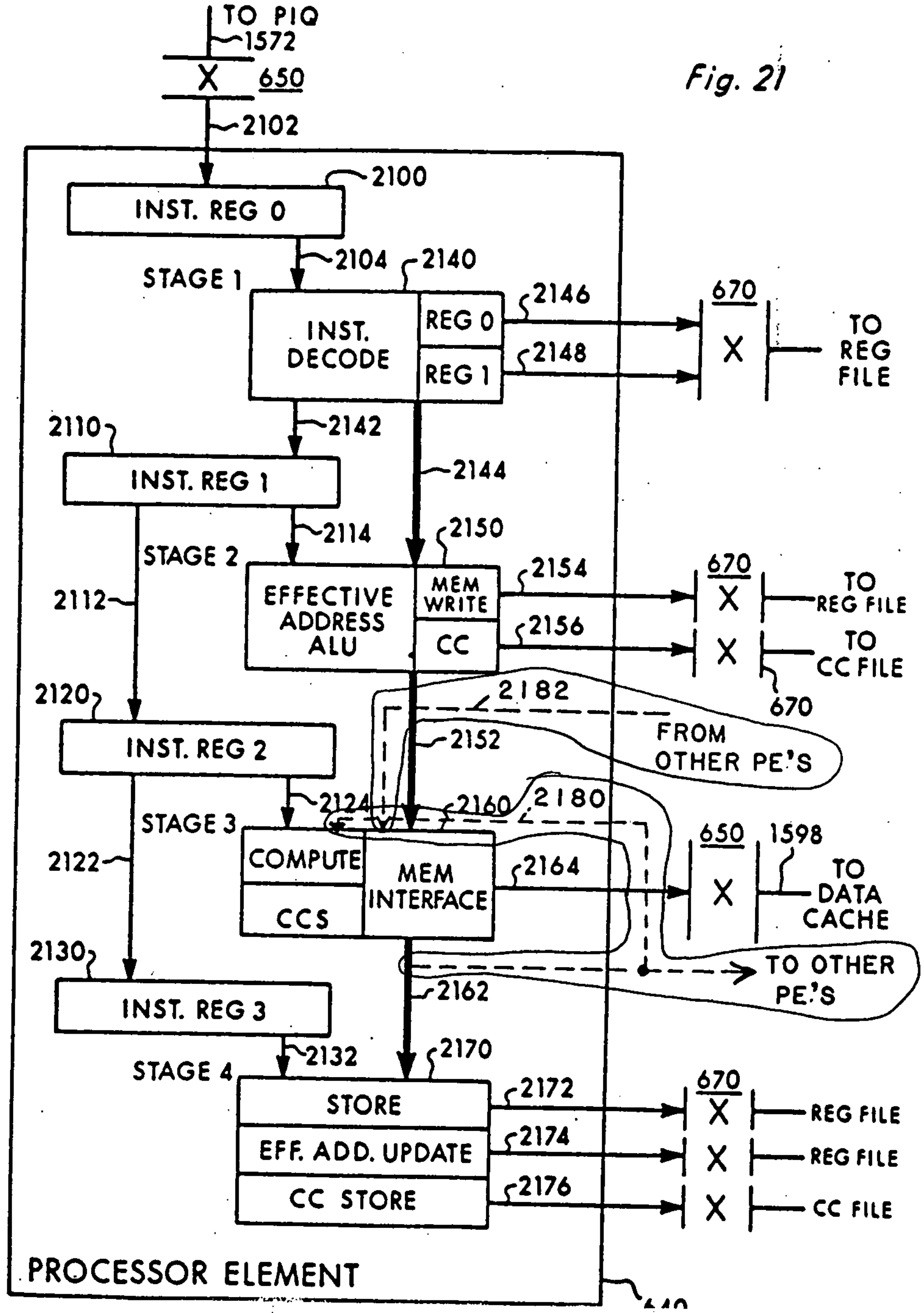

TO PIQ
1572
650
2102
2100
INST. REG 0
2104
2140
STAGE 1
INST. DECODE
REG 0
REG 1
2146
2148
670
TO REG FILE
2110
2142
2144
INST. REG 1
2114
2150
STAGE 2
2112
EFFECTIVE ADDRESS ALU
MEM WRITE
CC
2154
2156
670
TO REG FILE
TO CC FILE
670
2120
2182
INST. REG 2
2152
FROM OTHER PE'S
2124
2160
2180
STAGE 3
2122
COMPUTE
CCS
MEM INTERFACE
2164
650
1598
TO DATA CACHE
TO OTHER PE'S
2130
2162
INST. REG 3
2132
2170
STAGE 4
STORE
EFF. ADD. UPDATE
CC STORE
2172
2174
2176
670
REG FILE
REG FILE
CC FILE
PROCESSOR ELEMENT

Fig. 21

Fig. 22a

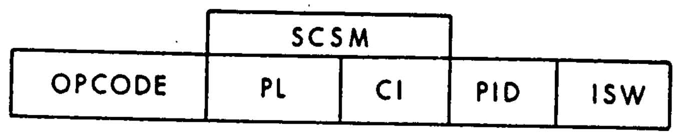


Fig. 22b

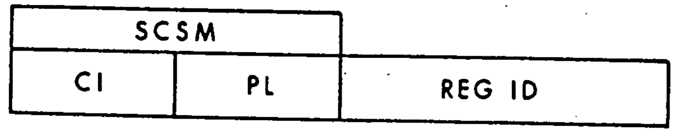


Fig. 22c

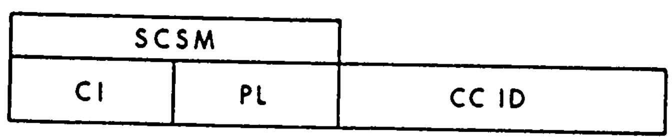
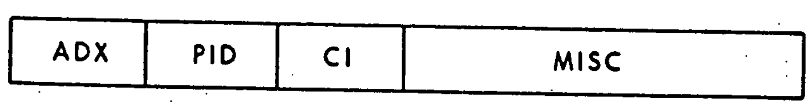


Fig. 22d



Part of Paper #6/C

Serial Number 11

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re: Application of :
Gordon Edward Morrison et al :
Serial No.: :
Filed: :
For: PARALLEL PROCESSOR SYSTEM AND PROCESSING NATURAL CONCURRENCIES AND METHOD THEREFOR :

PETITION PURSUANT TO 37 C.F.R. 1.102(d)
ADVANCEMENT OF EXAMINATION

Honorable Commissioner of Patents & Trademarks
Washington, D.C. 20231

Dear Sir:

Pursuant to the authority of 37 C.F.R. 1.102 and MPEP 708.01, Applicants respectfully petition the Commissioner of Patents and Trademarks to advance the examination of the above-identified application.

1. Fee - The fee of $60 is attached with this petition and any further fees should be charged to deposit account number 02-4980.

2. Pre-Examination Search - A pre-examination patentability search of Patent Office files was performed on this invention in a report dated December 21, 1983 by Ken Darnell, a professional searcher located in Arlington, Virginia. In addition, a Keyword search on Info-line was conducted. The search extended to the U.S. Patents categorized in U.S. classification Class 364, Subclass 200. In conducting the search, Examiners Chen, Springborn and Shaw were consulted.

The Washington Associate found the following references, copies of which are attached, to be of greatest pertinence:

Freiman, et al., (3,343,135), "Compiling Circuitry for a Highly-Parallel Computing System", September 19, 1967.
Culler, (3,771,141), "Data Processor With Parallel Operations for Instructions", November 6, 1973.
Gruner, (4,104,720), "CPU/Parallel Processor Interface with Microcode Extension", August 1, 1978.
Blum, et al, (4,109,311), "Instruction Execution Modification Mechanism for Time Slice Controlled Data Processor", Augsut 22, 1978.
Dennis, et al., (4,153,932), "Data Processing Apparatus for Highly Parallel Execution of Stored programs", May 8, 1979.
Kober, (4,181,936), "Data Exchange Processor for Distributed Computing System", January 1, 1980.
Bernhard, (4,228,495), "Multiprocessor Numerical Control System", October 14, 1980.
Gilliland, et al., (4,229,790), "Concurrent Task and Instruction Processor and Method", October 21, 1980.
Caril, (4,241,398), "Computer Network Line Protocol System", December 23, 1980.
Koehler, et al., (4,270,167), "Apparatus and Method for Cooperative and Concurrent Coprocessing of Digital Information", May 26, 1981.

The following references, copies of the Official Gazette notice are attached, were indicated to be of general interest to parallel processing task assignment:

3,470,540
4,496,551
4,573,852
4,060,849
4,126,893
4,130,865
4,247,892

And finally, the following references, copies of the Official Gazette notice are attached, were indicated to be of general interest only:

3,564,226
3,588,483
3,886,523
4,007,439
4,110,822
4,251,861
4,362,926
4,412,285
4,412,286

3. Literature and Patent Search Update - On February 12, 1985, a literature search was conducted in the DIALOG system. In that literature search, the following databases were specifically accessed:

1. INSPEC
2. DISSERTATION ABSTRACTS ON LINE
3. CLAIMS-U.S. PATENT ABSTRACTS WEEKLY
4. EI ENGINEERING MEETINGS

The search parameters were:

1. All references containing the root-word CONCURREN?
2. Of the references found in step 1, those further containing the root-words "SCALER?, PROCESS?, VECTOR?, or EXECUT?"
3. Of those references found in step 2, those further containing the root-words "SUPER?", ARCHITECT?, PARALLEL?, or MULTI?".
4. Of the references found in the third step, those further containing the root-words "INDEPEND? or DEPEND?."

Based upon the number of references uncovered in an individual database, either the first three steps listed above were used or all four steps. The clear limitation of this search is found in the requirement that each reference contained the root-word combinations: CONCURRENT? and INDEPEND? or CONCURRENT? and DEPEND?.

In addition, name searches were conducted on the following individuals:

1. VANAKEN, JERRY R.
2. ZICK, GREGORY L.
3. FISHER, JOSEPH L.
4. DESANTIS, ALFRED J.
5. FREIMAN, CHARLES V.
6. KUCK, D.J.

On February 21, 1985, an additional literature search on DIALOG was conducted solely by the undersigned. This search was conducted on the following databases:

1. CLAIMS-U.S. PATENT ABSTRACTS WEEKLY
2. INSPEC
3. DISSERTATION ABSTRACTS

In each of these databases, the search was conducted on the following individual, Charles E. McDowell. Furthermore, in the DISSERTATION ABSTRACTS database, the following search was conducted:

1. All references in descripter code classification of 0984 (the descripter code for the Charles E. McDowell dissertation reference).
2. All references containing the root-word "CONCURREN?" were excluded from those uncovered in step 1.
3. Of the references found in step 2, those which further contained the root-word "PARALLEL?."
4. Of the references uncovered in step 3, those which further contained the root-words "SCALER?, PROCESS?, VECTOR?, or ARCHITECT?"
5. The 172 references remaining in step 4 were then selectively culled by excluding those with the following non-pertinent root-words:

a.	FAULT?	k.	LABEL
b.	CONTOUR?	l.	RELAXATION
c.	COMMUNICA?	m.	CIPHER
d.	NEURAL?	n.	SPECTRUM
e.	SYNAPTIC?	o.	TRANSFORM
f.	RING?	p.	ELECTRO?
g.	AI	q.	MASK
h.	ARTIFICIAL	r.	LED
i.	INHERITANCE	s.	MONITOR?
j.	TOPOLOGY	t.	UNIX

On February 26, 1984, an additional literature search was conducted by the undersigned on the names of: Hagiwara, H.; Tomita, S.; and Shibayama, K. These persons were involved in the Japanese QA-1 project. No patents were found issued in their names. A large number of additional papers and speeches were uncovered in the EI MEETINGS and INSPEC databases. Based upon the descripter and identifier codes for the Hagiwara

references, a supplemental search in EI MEETINGS and INSPEC was conducted as follows:

1. All references in classification code C5220.
2. All references in step 1 further having the root-word PARALLEL? as a descripter and identifier.
3. All references in step 2 being published in the year 1984.

As a result, the following of all the references uncovered are believed to be the most pertinent:

Reigel, (3,611,306), "Mechanism to Control the Sequencing of Partially Ordered Instructions in a Parallel Data Processing System", October 5, 1971.

Lorie, et al., (4,435,758), "Method for Conditional Branch Execution in SIMD Vector Processors", March 6, 1984.

DeSantis, (4,468,736), "Mechanism for Creating Dependency Free Code For Multiple Processing Elements", August 28, 1984.

DeSantis, (4,466,061), "Concurrent Processing Elements For Using Dependency Free Code", August 14, 1984.

Hagiwara, H. et al., "A Dynamically Microprogrammable, Local Host Computer With Low-Level Parallelism", IEEE Transactions on Computers, C-29, No. 7, July, 1980, Pgs. 577-594.

Fisher, et al., "Microcode Compaction: Looking Backward and Looking Forward", National Computer Conference, 1981, Pgs. 95-102.

Fisher, et al., "Using an Oracle to Measure Potential Parallelism in Single Instruction Stream Programs", IEEE No. 0194-1895/81/0000/0171, 14th Annual Microprogramming Workshop, Sigmicro, October, 1981, Pgs. 171-182.

J.R. Vanaken, et al., "The Expression Processor, IEEE Transactions on Computers, C-30, No. 8, August, 1981, Pgs. 525-536.

Hagiwara, H. et al., "A User-Microprogrammable Local Host Computer With Low-Level Parallelism", Article, Association for Computing Machinery, #0149-7111/83/0000/0151, 1983, Pgs. 151-157.

McDowell, Charles Edward, "SIMAC: A Multiple ALU Computer", Dissertation Thesis, University of California, San Diego, 1983 (111 pages).

McDowell, Charles E., "A Simple Architecture for Low Level Parallelism", Proceedings of 1983 International Conference on Parallel Processing, Pgs. 472-477.

Fisher, A.T., "The VLIW Machine: A Multiprocessor for Compiling Scientific Code", Computer, 1984, Pgs. 45-52.

Fisher, et al., "Measuring the Parallelism Available for Very Long Instruction, Word Architectures", IEEE Transactions on Computers, Vol. C-33, No. 11, November, 1984, Pgs. 968-976.

Dennis, "Data Flow Supercomputers", Computer, November, 1980, Pgs. 48-56.

Bernhard, "Computing at the Speed Limit", IEEE Spectrum, July, 1982, Pgs. 26-31.
Davis, "Computer Architecture", IEEE Spectrum, November, 1983, Pgs. 94-99.
Requa, et al., "The Piecewise Data Flow Architecture: Architectual Concepts, IEEE Transactions on Computers, IEEE Transactions on Computers, Vol. C-32, No. 5, May, 1983, Pgs. 425-438.

In the two McDowell references, a parallel processing system utilizing low level parallelism was disclosed. McDowell differentiates between low level and high level parallelism in that low level parallelism pertains to the execution of two or more machine level operations in parallel whereas high level parallelism is the parallel execution of high level language constructs which includes source language statements, tasks, procedures, or entire programs.

Like the present invention, McDowell identifies and statically schedules low level parallel operations found in existing "sequential" programming languages. Further, he does not require the creation of special purpose programming languages or user modifications to the parallelism existing with the program at compile time, rather he simply analyzes the low level parallelism existing within the basic blocks (BBs) that make up the program.

In the McDowell SIMAC system, processor elements are identical and communicate through a shared memory and a set of shared registers. The processor elements are controlled by a control processor that is fed by a single instruction stream.

McDowell recognizes his particular parallel processing system does not fit into the standard categories of: homogeneous multi-processors, non-homogeneous multi-processors, array processors, and pipeline processors. Rather, McDowell classified his computer system in the class of Schedulable Parallel Instruction Execution (SPIE) computers. McDowell's

reasoning as to why SIMAC is categorized as a SPIE processor is as follows:

> SPIE processors have only one instruction stream and are therefore immediately eliminated from the first two groups. They have some of the characteristics of array processors, but they are distinctly different in that each of the several processing elements may be executing different operations. In array processors all processing elements are restricted to executing the same instruction on different data. SPIE processors are also similar in capability to pipelined machines but again they are distinct in that each processing element may be given a new operation each instruction cycle that is independent of the operation it was performing in the previous cycle. Pipelined machines only specify a single operation per instruction cycle, and then other operations on other processing elements may be triggered from that one initial operation. Dissertation Thesis at 11.

The present invention also is properly categorized as a SPIE processing system since each processor element may be given a new operation each instruction cycle that is independent of the operation it was performing in the previous cycle. Hence, the McDowell reference provides important background material to the teachings of the present invention. However, important distinctions exist between McDowell's SIMAC system and the system of the present invention. SIMAC is quoted as being a SPIE machine that can perform scalar concurrent execution of a single user's program (i.e., SIMD). The present invention also fits into this SIMD category, however, it is further capable of performing scalar concurrent execution for multiple instruction streams (i.e., MIMD) and further for single or multiple context (i.e., SC-MIMD or MC-MIMD).

SIMAC implicitly assigns instructions (termed "machine operations") to processor elements by virtue of the physical ordering of the instructions when they are grouped together into "parallel instructions." The identity of the assigned processor is not a part of the instruction per se. The present invention teaches that the order of the instructions within a

group of concurrently executable instructions does not determine the processor assignment. Rather, the processor assignment, under the teachings of the present invention, is explicitly made by adding a logical processor number to become physically part of the instruction which is performed in software prior to execution. In addition, the present invention distinguishes the logical processor number from the actual physical processor number that executes the instruction. This allows for further machine independence of the software in that the physical processor assignment is performed dynamically at execution time.

SIMAC code generation can be divided into three components during which the static allocation and scheduling of its instructions are performed (prior to execution). First, is the determination of the machine operations to be performed. This is the typical code generation phase of a compiler. Second, is the formation of these operations into parallel instructions containing the operations that can be done in parallel. The third component comprises the ordering (scheduling) of the parallel instructions so

> "...to allow the execution of two [parallel instructions] containing different length [machine operations] to overlap using only simple hardware controls..."
> McDowell's Conference Paper @ Pg. 475.

In other words, McDowell attempts to schedule instructions with differing length execution times so that they can execute concurrently with a minimum amount of hardware support. To the contrary, the present invention does not require this third component. The use of the logical processor number and the instruction firing time that are physically attached to each instruction preclude the need for this phase.

As stated above, SIMAC performs its concurrency detection during compile time only. As a result, the static analysis of the concurrency present within the instruction stream is language dependent. In contrast, the concurrency detection of present invention is language independent since the detection takes place after compilation and prior to execution. The present invention can perform its concurrency detection and scheduling without requiring any modifications to the compiler.

Like SIMAC, the present invention is designed to be independent of the number of processor elements contained within the system. The processor elements in SIMAC, however, are conventional load/store RISC style processors that contain contextual information regarding the state of the previous execution status. These are the "T" and "V" bits associated with each of the SIMAC processor elements. These are used for branching and are the result of executing certain comparison and test instructions. The processor elements of the present invention are context free processors that support a RISC-like instruction set. The term "context free" means that the processor elements contain no state information, e.g., condition codes, registers, program status words, flags, etc. Like SIMAC, the processor elements of the present invention contain no local registers. All operations are performed on data already stored in an array of registers and are accessible by any of the individual processor elements through a register switch.

In SIMAC, only a single set (or level) of 32 registers is available and is shared by all processor elements. Each register may be physically read by any or all processor elements simultaneously. However, only one processor element

may write into a specific register at any one time. (Dissertation Thesis at 26) Whereas the registers of this invention also have this characteristic, the architecture of the register set in the present invention is much different than that of the SIMAC machine. The SIMAC machine contains a single level of registers. The present invention contains a number of levels of register sets; each level corresponding to a procedural depth relative to the main program. In addition, each set of levels is replicated to support each context or user that is currently being executed. Thus, under the teachings of the present invention, contexts may be switched or procedures entered without having to flush any registers to memory.

SIMAC does not discuss the need for the production and assignment of condition codes. Without the use of multiple condition code sets, it is not possible to distribute instructions which affect the condition codes across multiple processor elements and guarantee proper condition code data for the execution of subsequent dependent instructions. The present invention teaches a method for the management and assignment of multiple condition code sets. The management and assignment of this resource is analogous to that done for the register resources.

SIMAC also requires a separate control processor that performs the branching operations. This processor contains the program counter and executes the branch operations Each processor element in SIMAC contains three status bits used in the branching operations. The present invention does not use a control processor, nor does it use a program counter and its processor elements contain no such status bits. SIMAC is a centralized control system under the Davis classification whereas the present invention is decentralized.

Hence, the present invention is significantly more general than the SIMAC approach since the present invention performs inter-program parallelism as well as intra-program parallelism.

In addition, while SIMAC provides a register array, McDowell indicates that this, perhaps is not desirable because of cost and proposes that future work be done on an arrangement other than shared registers (Id. at 79) The present invention not only makes use of the shared register concept but also provides sharing of condition codes and the management of those condition codes as well. While McDowell, implicitly, assigns the machine operations to his processor elements, the present invention explicitly extends each instruction with a logical processor number as well as providing a specific firing time for the instruction. There is no disclosure in SIMAC of either a firing time or a logical processor number. Therefore, the master controller, in SIMAC, uses a program counter that provides the control for the processor elements. Under the teaching of the present invention, a set of logical resource drivers (one for each context or user) is provided and instruction selection is based upon the firing times added to each instruction within a basic block (i.e., time driven).

DeSantis sets forth in U.S. Letters Patent Nos. 4,466,061 and 4,468,736 a concurrent data processor which is:

> adapted to receive strings of object code, form them into higher level tasks and to determine sequences of such tasks which are logically independent so that they may be separately executed ('736 patent at col. 2, lines 50-54).

The logically independent sequences are separately executed by a plurality of processor elements. The first step in the DeSantis approach is to hardware compile the program into strings of object code or machine language in a form which is particularly designed for the computer architecture to provide a dependency free code. The DeSantis invention then forms an

independency queue so that all processing for the entire queue is accomplished in one step or cycle. The hardware structure for the DeSantis approach is set forth in Figure 2 of the patent and contemplates the use of a number of small processor elements (SPEs) each with local memories.

The nature of the interconnection between the local memories of each SPE and the direct storage module 14 is not clear. The patent states:

> In the meantime, respective data items required for execution have been fetched from main memory and stored at the appropriate locations in local memories which locations are accessed by the pointers and the job queue ('736 patent at Col. 6, Lines 19-23).

It does not appear from this description, that each individual SPE has access to the local memories of the other SPE rather, this allocation seems to be made by the direct storage module (DSM). Hence, under DeSantis, the queue which contains the independent sequences for processing delivers the string to an identified processor element, so that the processor elements can process all strings concurrently and in parallel, the direct storage module must deliver the necessary results into the local memories for each SPE. With respect to each individual SPE, DeSantis states:

> Since the respective processors are provided to execute different functions, they can also be special purpose microprocessors containing only that amount of logic circuitry required to perform the respective functions. The respective circuits are the arithmetic logic unit, shift unit, multiply unit, indexing unit, string processor and decode unit ('736 patent at Col. 7, Lines 7-13).

The primary difference between the approach of the present invention and DeSantis relates to when the detection of the concurrency occurs. TDA performs its concurrency detection during a pre-processing stage at the object level using information normally thrown away by the source code level

language processors. DeSantis performs it with part of the hardware collectively called the cache mechanism 10 (see Figure 1) dynamically during execution time. Hence, the DeSantis approach constantly uses overhead and resources to hardware de-compile each program. A secondary difference relates to the execution of individual streams of dependent instructions. DeSantis requires that a stream of dependent instructions be executed on the same processor. To the contrary, the present invention permits the execution of a stream of dependent instructions on the same or multiple processor elements.

It is believed that the aforesaid McDowell and DeSantis references are the most pertinent of the references discovered in the patentability investigation and, therefore, more discussion occurs for it than the following references.

J.R. Vanaken in "The Expression Processor" article classifies his processor as a "direct descendant of the tree processor." There is a similarity between the Vanaken architecture and the architecture of the present invention wherein each processor element is capable of accessing, through an alignment network, any one of a number of registers located in a register array. Vanaken utilizes thirty-two processor elements and eight register modules in the register storage. A crossbar switch is disclosed for the alignment network. In this configuration, the identical processor elements of Vanaken do not exchange data directly with the main memory. Rather, data transfers take place between the register storage and the memory. This is not the configuration of the present invention. Vanaken further contemplates utilizing a separate compiler (only for scaler programs) for the detection of low level parallelism and for the assignment of detected parallelism to individual processor elements. How this is

done, however, is not disclosed. Vanaken simply discloses the desired goals:

> First, it [the compiler] must detect the potential parallelism in the program. Second, it must map detected parallelism onto the structure of the target machine. The Expression Processor at 535.

Vanaken refers only to the parallelism detection techniques presented by Kuck. However, Vanaken's tree structured processor element architecture requires that the computational task be modeled as a computational tree, or as a computational wavefront (wavefront ref: Kuck's work). In the former case, processor assignment is accomplished by assigning a processor to each node in the tree. If there are more nodes in the tree than processors, the task must be broken down into smaller subtasks that will fit on the processor array. In the latter case, the wavefront propagates from the lowest level of processors to the next higher level and so on until the highest level is achieved. Again, if the number of processors required by the wavefront is greater than the number of processors available, the task must be broken into smaller subtasks. This type of flow through ordered processors does not exist in the present invention.

In the Hagiwara articles, the disclosed MIMD QA-1 computer system appears to be a very long instruction word (VLIW) machine having a 160 bit or 80 bit word. The Hagiwara references term this a "horizontal-type microinstruction" and the QA-1 machine takes advantage of the low-level parallel concurrency, at the microcode level. The QA-2 system has 62 working registers constructed of large capacity/high-speed RAM chips interconnected over a network to each of the four processor elements. The earlier QA-1 system utilized a stack of only 15 working registers. Hence, the QA-2 hardware design

contemplates the use of individual processor elements having full access to a number of working registers through a switching network. In QA-2, through use of the network, the results of one processor element are delivered to a working register which is then delivered to another processor element as input data. Hence, the QA-1 and QA-2 configurations take advantage of scaler concurrency, multiple ALUs, and the sharing of working registers. The QA-2 approach modifies the QA-1 approach by providing special purpose registers in the register file (i.e., constant, general, indirect, and special); such a division, however, is not apparent in the QA-1 architecture. From a compiling point of view, the QA-1 was primarily designed to be encoded by a programmer at the microcode level to take programming advantage of the inherent power of the system's architecture. The QA-2 machine which is directed towards the personal computer or local host computer (e.g., for laboratory use) contemplates that programs written in high-level languages will be compiled into the QA-2 microprogrammed interpretations.

The four Fisher references disclose a SIMD system based upon a "very long instruction word" (VLIW). The goal presented in these articles is to determine the low level parallelism or "fine-grained parallelism" of a program in a separate compiler so that all individual movements of the data within the VLIW machine are completely specified at compile time. At the outset, it is important to recognize that the hardware architecture for implementing Fisher's VLIW machine is not discussed in these references other than in a general sense. Fisher contemplates:

> Each of the eight processors contain several functional units, all capable of initiating an operation in each cycle. Our best guess is two integer ALUs, one pipelined floating ALU, one memory port, several register banks, and a limited crossbar

for all of these to talk to each other. Computer article at pg. 50.

Rather, the Fisher articles concentrate on the software compiling technique called "trace scheduling" and nicknamed BULLDOG. Fisher further requires an instruction word of "fixed length." The length of Fisher's instruction word, therefore, dictates the amount of hardware required to process the instruction word. In that respect, the present invention is much more dynamic and fluid since it can have any number of processors wherein each processor processes the parallel instruction during the instruction firing time.

Lorie et al. (U.S. Letters Patent No. 4,435,758) sets forth a technique for ordering instructions for parallel execution during compile time by adding code to the instruction stream. The present invention does not add code to the instruction stream.

The Reigel patent (U.S. Letters Patent No. 3,611,306) relates to the ordering of the detected parallel instructions for a particular computer architectural approach.

The patent issued to Freiman, et al. (U.S. Letters Patent No. 3,343,135) teaches a dynamic compiling system, hardware based, for use in a multiprocessor environment for analyzing particular types of mathematical expressions for parallel execution opportunities and then to automatically assign individual operations to processors within the system. Freiman first identifies the concurrencies, in the hardware, and determines the time periods within which the operations may be performed. These determined times are "the earliest times in which a given operation may be performed." The Freiman invention also analyzes the particular mathematical expression and determines, in the hardware, "the latest times in which an

operation can be performed." This information generated from the analysis of the mathematical algorithm is stored in a word register which is then used in the multiprocessor environment. Each processor has associated with it a processor control which for all processors is identical and which is driven by six control fields. One function of the control block is to determine whether or not a particular processor is ready to receive a job, to indicate when it has received a job, and to analyze each job to see if the processor can proceed immediately with the job or whether it must wait until one of the operants of the job is already assigned. Once a processor is assigned to a task the necessary data contained in the results register may not be available at that particular time and hence the processor must wait until a test of the results register indicates that the result is available. Once the result is available, the processor will perform its operation and place its result in the appropriate result register for access by another processor. All processors in Freiman have access to the results register by means of a crossbar switch.

The patent issued to Dennis et al. (U.S. Letters Patent No. 4,153,932) teaches a highly parallel multiprocessor for the concurrent processing of programs represented in data flow form. Specifically, a mathematical computation as shown at column 5, lines 17-23 and as represented in Figure 2 in data flow language is used as an example. Hence, the data flow form is generated and stored in the memory of the processor in designated instruction cells wherein each instruction cell corresponds to an operator in the data flow program. In operation, when an instruction cell is complete (i.e., containing the instruction and all necessary operands), a signal is generated to the Arbitration Network which directs

the flow of the information in the instruction cell to the identified processor. The processor operates on the information and delivers it back through a distribution network into the main memory.

The patent issued to Culler (U.S. Letters Patent No. 3,771,141) discloses a high speed processor capable of parallel operations on data. Multiple or parallel processors, however, are not used. Rather, the parallelism is achieved structurally by implementing each of the processor's data registers with four separate multi-bit data input ports. Hence, four operations can be performed at once. This results in a "tightly coupled" arrangement since the data from any given processor register is ready to be received by any other register at the next clock cycle. The control of which register is to receive which information occurs at the object code level and, as shown in Table 1 of Culler, four additional fields are provided in each object code instruction for controlling the flow of data among the processor's registers. Culler states:

> The data pad memory is tightly coupled to the arithmetic unit and serves as an effective buffer between the high speed arithmetic unit and a large capacity randam access core memory. As an indication of the effective speed, a complete multiplication of two signed eight bit words can be accomplished in three cycles or 375 nanoseconds.

The Gilliland et al. patent (U.S. Letters Patent No. 4,229,790) sets forth a processor for processing different and independent jobs concurrently through the use of pipelining with precedent constraints. The Gilliland invention is capable of processing programming tasks concurrently as well as processing the parallel parts of each programming task concurrently. It appears that the invention operates on concurrencies in the program at the source code level rather

than at the object code level. The Gilliland processor is capable of processing up to 128 parallel processor paths.

The patent issued to Blum et al. (U.S. Letters Patent No. 4,109,311) relates to a data processing system based upon time slice multiprogramming. The Blum invention contemplates a conventional multiprocessor arrangement wherein a first processor controls a keyboard and display interface, a second processor controls printer, disk storage, and tape storage interface, and a third processor executes the problem oriented programs located in the main storage.

Gruner (U.S. Letters Patent No. 4,104,720) sets forth a CPU control multiprocessor system wherein each parallel processor is directly interfaced through an interface control circuit to the CPU. Each interface circuit contains memory for storing portions of the micro instructions contained within the CPU. Gruner calls this an "extension" of the micro instructions from the CPU into the interface circuit for each parallel processor. The Gruner system while controlling each interface circuit to allocate and to assign concurrent tasks utilizes a conventional bus structure for the transfer of information between the processors.

The patent issued to Caril (U.S. Letters Patent No. 4,241,398) relates to a supervisory control system wherein a central processing unit controls and supervises the operation of a number of remote processing units. The Caril invention relates to a low overhead line protocol for controlling the asynchronous exchange of communication information between the central processing unit and each remote processing unit.

Bernhard et al. (U.S. Letters Patent No. 4,228,495) relates to a multiprocessor numerical control system. The Bernhard invention relates to a main processor coupled over a bus structure to a plurality of programmable interface processors.

Koehler et al. patent (U.S. Letters Patent No. 4,270,167) discloses a multiprocessor arrangement wherein a central processor has primary control and access to a local bus and wherein the remaining plurality of processors also share access to the local bus. Arbitration among the processors is provided to the local bus as well as the generation of query status and processor status signals.

The patent issued to Kober (U.S. Letters Patent No. 4,181,936) relates to a distributed computing system for increasing the efficiency of the data bus.

The Requa article discusses a piecewise data flow architecture which seeks to combine the strengths of other supercomputers. The goal of the Requa architecture is to take the source code program and recompile it into "basic blocks." These basic blocks are designed to be no longer than 255 instructions and the natural concurrencies existing in each basic block are identified and processed separately by a number of "scaler processors." The natural concurrencies are encoded as part of the instructions to the scaler processors. The concurrent instructions waiting for execution reside in a stack of registers to which all of the scaler processors has access to deliver data into and to receive data from. As in the Dennis approach, when a particular instruction residing in the register is completed (i.e., when a prior data dependency has been satisfied and stored) the scaler instruction apparently raises a flag which will be detected for delivery to a scaler processor.

In the Bernhard article, the author surveys the current state of art concerning supercomputer design. The article discusses the current supercomputer research projects and provides critical comments concerning each type of design. The

disclosure in this article appears to be cumulative to the analysis set forth above and none of the approaches set forth suggest the TDA processor register communication technique.

4. Applicant's Invention - Because of the large number of claims attached to the patent application, it is believed helpful to the Examiner to index the claims.

1. Multicontext - Multiple Instruction Stream - Multidata (MC-MIMD) APPARATUS Claims 1-10
2. Single Context Multiple Instruction Stream-Multiple Data (SC-MIMD) APPARATUS Claims 11-14
3. MC-MIMD METHOD CLAIMS 15-21
4. SC-MIMD METHOD CLAIMS 22-39
5. MC-MIMD METHOD CLAIMS 41-52
6. SC-MIMD APPARATUS CLAIMS 53-68
7. Program Level Technique (Figures 13, 14) Claims 69-70
8. Performance of Branch Operation (Figures 1, 6, 19) - Claims 71-72
9. Context Free Processing Claim 73

The present invention provides both a method and a system for a non-von Neuman computer that is primarily adaptable to either single or multiple context MIMD configurations, but also to SIMD and SISD configurations. The method and system of the present invention is further operative upon a number of conventional programs without user intervention.

The present invention statically determines at a very fine level of granularity, the natural concurrencies in the basic blocks of programs at essentially the object code level and adds a processor and timing information to each instruction in each basic block in order to provide a time driven decentralized control. This feature of determining

natural concurrencies in basic blocks and adding processor and timing information to each instruction is not set forth in any of the above cited references and is specifically claimed in the following independent claims: 1, 2, 3, 6, 7, 10, 11, 12, 13, 15, 18, 21, 22, 23, 26, 27, 36, 39, 40, 50, 53, 60 and 68. The detection and low-level scheduling of the natural concurrencies and the adding of the intelligence occurs only once for a given program after conventional piling and prior to execution. At this time all instruction resources are determined.

The present invention further executes the basic blocks containing the added intelligence on a system using a plurality of identical processor elements each of which is incapable of retaining execution state information from prior operations. Hence, all processor elements are context free and this feature is not found in any of the above references and is specifically claimed in the following independent claims: 1, 2, 3, 6, 11, 13, 60, 68, and 73. A given processor element in the present invention is capable of executing an instruction from one user followed from an instruction from another user or followed by an instruction from another independent concurrent stream of the same user. All context information necessary for processing a given instruction being contained elsewhere in the system.

In addition, the particular approach set forth in Figures 13 and 14 and claimed in independent Claims 69 and 70 relating to the adding of program level information to each instruction and providing different sets of registers for each level is not set forth in any of the above references and this particular feature as found in these claims is believed to be patentable over the art.

Finally, the particular approach to handling branch instruction as set forth in Figures 1, 6, and 19 and as positively claimed in Claims 71 and 72 are also not set forth in any of the above prior art references and are patentable thereover. These claimed features, set forth above, are not shown, or even suggested, by the prior art.

5. Copies - One copy of each of the above-cited references is included herewith.

This application is therefore considered to be in condition for allowance and expedited examination and allowance of this application is respectfully requested.

Respectfully submitted,

BURTON & DORR

Date: Oct 31, 1985 By: Robert C Dorr

Robert C. Dorr - #27,782
1240 Century Tower No. 1
720 South Colorado Boulevard
Denver, Colorado 80222
(303) 759-3950